

Package ‘risk.assessr’

April 8, 2026

Title Assessing Package Risk Metrics

Version 3.1.1

Description A reliable and validated tool that captures detailed risk metrics such as R 'CMD' check, test coverage, traceability matrix, documentation, dependencies, reverse dependencies, suggested dependency analysis, repository data, and enhanced reporting for R packages that are local or stored on remote repositories such as GitHub, CRAN, and Bioconductor.

License GPL (>= 2)

URL <https://sanofi-public.github.io/risk.assessr/>

BugReports <https://github.com/Sanofi-Public/risk.assessr/issues>

Depends R (>= 4.1.0)

Imports remotes, callr, checkmate, covr, desc, dplyr, fs, methods, pkgload, purrr, rmarkdown, rcmdcheck, rlang, xml2, stringr (>= 1.5.0), tidyr, utils, curl, jsonlite, memoise, testthat (>= 3.0.0), BiocManager, glue

Suggests devtools, forcats, here, htmltools, htmlwidgets, kableExtra (>= 1.4.0), knitr, magrittr, openxlsx, R6, S7, readr, roxygen2, testit, tibble, tidyselect, tools, withr, mockery, ggplot2, DT

Encoding UTF-8

RoxygenNote 7.3.3

Config/testthat/edition 3

Config/build/clean-inst-doc false

VignetteBuilder knitr

NeedsCompilation no

Author Edward Gillian [cre, aut] (ORCID:

<<https://orcid.org/0000-0003-2732-5107>>),

Hugo Bottois [aut] (ORCID: <<https://orcid.org/0000-0003-4674-0875>>),

Paulin Charliquart [aut],

Andre Couturier [aut],

Sanofi [cph, fnd]

Maintainer Edward Gillian <edward.gillian-ext@sanofi.com>

Repository CRAN

Date/Publication 2026-04-08 19:50:02 UTC

Contents

assess_pkg_r_package	3
average_issue_close_time	4
bioconductor_reverse_deps	5
build_dependency_tree	6
check_and_fetch_cran_package	7
check_cran_package	8
check_forbidden_notes	8
check_suggested_exp_funcs	9
contains_vignette_folder	10
count_commits_last_months	11
cran_packages	11
cran_revdep	12
create_coverage_skip_stf	13
create_traceability_matrix	14
dependsOnPkgs	16
detect_version_conflicts	17
download_and_parse_dependencies	17
extract_package_version	18
extract_thresholds_by_id	19
extract_thresholds_by_key	20
extract_version	20
fetch_all_dependencies	21
fetch_bioconductor_package_info	22
fetch_bioconductor_releases	22
generate_html_report	23
generate_traceability_matrix	24
get_bioconductor_package_url	25
get_commits_since	26
get_cran_package_url	27
get_cran_total_downloads	28
get_exports	29
get_github_data	30
get_host_package	31
get_internal_package_url	32
get_package_download_bioconductor	32
get_package_download_cran	33
get_package_tarfile	34
get_pkg_name	35
get_pubmed_by_year	36
get_pubmed_count	37
get_risk_analysis	37

get_risk_definition	38
get_session_dependencies	39
get_suggested_exp_funcs	40
get_versions	40
install_package_local	41
is_base	42
list_badges	43
map_test	43
map_tests_stf	44
modify_description_file	45
parse_bioconductor_releases	45
parse_package_info	46
print_tree	47
risk_assess_pkg	48
risk_assess_pkg_lock_files	49
run_covr_modes	49
run_covr_skip_nstf	51
run_covr_skip_stf	52
set_up_pkg	53
write_summary_report	54

Index **56**

assess_pkg_r_package *Assess an R package by name (deprecated)*

Description

Use [risk_assess_pkg()] with ‘package’ instead.

Usage

```
assess_pkg_r_package(package_name, version = NA, repos = getOption("repos"))
```

Arguments

- package_name Package name.
- version Package version. Default ‘NA’ uses latest.
- repos Repository URLs. Default ‘getOption("repos”)’.

Value

Assessment results or ‘NULL’.

`average_issue_close_time`*Calculate Average Time to Close GitHub Issues*

Description

This function uses the GitHub API to retrieve closed issues from a repository and calculates the average time (in hours) it took to close them, excluding pull requests.

Usage

```
average_issue_close_time(  
  owner,  
  repo,  
  max_pages = 10,  
  per_page = 100,  
  headers = c(Accept = "application/vnd.github.v3+json")  
)
```

Arguments

<code>owner</code>	Character. GitHub organization or username.
<code>repo</code>	Character. Repository name.
<code>max_pages</code>	Integer. Maximum number of API result pages to query. Each page can contain up to 'per_page' issues. Default is 10.
<code>per_page</code>	Integer. Number of issues to request per page (max 100). Default is 100.
<code>headers</code>	Named character vector of HTTP headers to send with the API request. Default sets GitHub API version.

Value

Numeric. Average time in hours to close issues, or 'NA' if no closed issues were found.

Examples

```
## Not run:  
average_issue_close_time("tidyverse", "ggplot2")  
  
## End(Not run)
```

`bioconductor_reverse_deps`*Find Bioconductor Package Reverse Dependencies*

Description

This function returns the reverse dependencies for a given Bioconductor package.

Usage

```
bioconductor_reverse_deps(  
  pkg,  
  which = "Imports",  
  only.bioc = FALSE,  
  version = BiocManager::version(),  
  db = NULL,  
  biocdb = NULL  
)
```

Arguments

<code>pkg</code>	Character string. The name of the package for which to find reverse dependencies.
<code>which</code>	Character vector. The dependency categories to check. One or more of "Depends", "Imports", "LinkingTo", "Suggests", or "Enhances". Defaults to "Imports".
<code>only.bioc</code>	Logical. If TRUE (default), only reverse dependencies that are Bioconductor packages are returned.
<code>version</code>	Bioconductor version to use. Defaults to the current version.
<code>db</code>	Optional. A pre-loaded package database to use for lookups.
<code>biocdb</code>	Optional. A pre-loaded Bioconductor package database.

Value

A named list of reverse dependency package names.

Examples

```
## Not run:  
# Get reverse Imports dependencies as a list:  
bioconductor_reverse_deps("limma")  
  
# Get multiple categories:  
bioconductor_reverse_deps("limma", which = c("Depends", "Suggests"))  
  
## End(Not run)
```

build_dependency_tree *Build a Dependency Tree for an R Package*

Description

This function constructs a nested list representing the dependency tree of a specified R package. The recursion stops at the specified maximum depth level (default: 3) or until a base package is reached.

Usage

```
build_dependency_tree(  
  package_name,  
  level = 1,  
  get_license = FALSE,  
  max_level = 3  
)
```

Arguments

package_name	A character string specifying the name of the package.
level	An integer indicating the current recursion depth (default: 1).
get_license	A logical value indicating whether to extract and include license information from DESCRIPTION files (default: 'FALSE').
max_level	An integer specifying the maximum depth level for dependency recursion (default: 3). Set to a higher value to explore deeper dependency layers.

Value

A nested list representing the dependency tree, where base packages are marked as "base". If 'get_license' is 'TRUE', each package entry in the tree includes a 'license' field containing that specific package's license (extracted from its DESCRIPTION file). Each package's license is independent - the license of a parent package does not imply the licenses of its dependencies.

Examples

```
## Not run:  
build_dependency_tree("ggplot2")  
build_dependency_tree("ggplot2", get_license = TRUE)  
build_dependency_tree("ggplot2", max_level = 5) # Deeper dependency exploration  
  
## End(Not run)
```

`check_and_fetch_cran_package`*Check and Fetch CRAN Package*

Description

This function checks if a package exists on CRAN and retrieves the corresponding package URL and version details. If a specific version is not provided, the latest version is used.

Usage

```
check_and_fetch_cran_package(package_name, package_version = NULL)
```

Arguments

`package_name` A character string specifying the name of the package to check and fetch.

`package_version` An optional character string specifying the version of the package to fetch. Defaults to 'NULL'.

Value

A list with one of the following structures:

package_url Character string; the URL to download the package tarball.

last_version A named list with `version` and `date` for the latest available version.

version Character string; the requested version (or NULL if not specified).

all_versions A list of named lists, each with `version` and `date`, representing all available versions.

Examples

```
## Not run:  
# Check and fetch a specific version of "ggplot2"  
result <- check_and_fetch_cran_package("ggplot2", package_version = "3.3.5")  
print(result)  
  
## End(Not run)
```

check_cran_package *Check if a Package Exists on CRAN*

Description

This function checks if a given package is available on CRAN.

Usage

```
check_cran_package(package_name)
```

Arguments

package_name A character string specifying the name of the package to check.

Value

A logical value: 'TRUE' if the package exists on CRAN, 'FALSE' otherwise.

Examples

```
## Not run:  
# Check if the package "ggplot2" exists on CRAN  
check_cran_package("ggplot2")  
  
## End(Not run)
```

check_forbidden_notes *Reclassify Forbidden Notes as Errors in rcmdcheck Results*

Description

This internal helper function scans the 'notes' field of an 'rcmdcheck' result object for specific patterns that indicate more serious issues. If any of these patterns are found, the corresponding notes are reclassified as errors by moving them from the 'notes' field to the 'errors' field.

Usage

```
check_forbidden_notes(res_check, pkg_name)
```

Arguments

res_check A list representing the result of an 'rcmdcheck' run. It should contain at least the elements 'notes', 'warnings', and 'errors', each being a character vector.

pkg_name name of the package

Value

A modified version of 'res_check' where certain notes matching forbidden patterns are moved to the 'errors' field.

Examples

```
pkg_name <- "foo"
res_check <- list(
  notes = c("'foo' import not declared from: 'bar'",
            "Namespace in Imports field not imported from: 'baz'",
            "no visible global function definition for 'qux'",
            "some harmless note"),
  warnings = character(0),
  errors = character(0)
)
updated <- check_forbidden_notes(res_check, pkg_name)
print(updated$errors) # Should include the first three notes
print(updated$notes) # Should include only the harmless note
```

check_suggested_exp_funcs

Function to check suggested exported functions

Description

This function checks the exported functions of target package against the exported functions of Suggested dependencies to see if any Suggested packages should be in Imports in the DESCRIPTION file

Usage

```
check_suggested_exp_funcs(pkg_name, pkg_source_path, suggested_deps)
```

Arguments

pkg_name - name of the target package
pkg_source_path - source of the target package
suggested_deps - dependencies in Suggests

Value

- data frame with results of Suggests check

`contains_vignette_folder`*Check for Vignette Folder and .Rmd Files in a .tar File*

Description

This function checks if a given .tar file contains a 'vignettes' folder and if there is at least one .Rmd file within that folder. If both 'vignettes' and 'inst/doc' folders exist, the function will return FALSE.

Usage

```
contains_vignette_folder(tar_file)
```

Arguments

`tar_file` A character string specifying the path to the .tar file to be checked.

Details

The function checks if the specified file exists and has a valid .tar extension using `utils::untar`. If the file is empty or any error occurs during the extraction, the function stops and returns an error message. If both 'vignettes' and 'inst/doc' folders exist, the function returns FALSE. If the 'vignettes' folder exists and contains at least one .Rmd file, the function returns TRUE. Otherwise, it returns FALSE.

Value

A logical value: TRUE if the 'vignettes' folder exists and contains at least one .Rmd file, and neither 'vignettes' nor 'inst/doc' folders are present, FALSE otherwise.

Examples

```
## Not run:
tar_file <- system.file("test-data", "here-1.0.1.tar.gz",
  package = "risk.assessr")
result <- contains_vignette_folder(tar_file)
print(result)

## End(Not run)
```

 count_commits_last_months

Count Commits in the Last Months

Description

Counts the total number of commits from the last ‘months‘ months using a data frame with weekly commit data (column ‘week_start‘).

Usage

```
count_commits_last_months(df, months = 1, today = Sys.Date())
```

Arguments

df	A data frame with ‘week_start‘ (Date) and ‘n_commits‘ (int) columns.
months	Integer. Number of months to look back (default is 1).
today	today’s date.default is Sys.Date()

Value

Integer. Total number of commits in the time window.

Examples

```
## Not run:
df <- get_commits_since("tidyverse", "ggplot2", years = 1)
count_commits_last_months(df, months = 3)

## End(Not run)
```

 cran_packages

Retrieve the List of CRAN Packages (Internal)

Description

Downloads and caches the current list of packages available on CRAN.

Usage

```
cran_packages(cache = TRUE)
```

Arguments

cache	Logical. If TRUE (default), the result is cached using <code>memoise::memoise()</code> with a timeout of 30 minutes to avoid repeated downloads. If FALSE, the metadata is fetched directly from CRAN each time the function is called.
-------	---

Details

The function downloads the `packages.rds` file from the CRAN website, which contains metadata about all available packages. The result is cached using `memoise::memoise()` with a timeout of 30 minutes.

Value

A matrix of package metadata similar to the output of `utils::available.packages()`, with package names as row names.

Examples

```
pkgs <- cran_packages()
head(rownames(pkgs))
```

cran_revdep

Find Reverse Dependencies of a CRAN Package

Description

This function finds the reverse dependencies of a CRAN package, i.e., packages that depend on the specified package.

Usage

```
cran_revdep(
  pkg,
  dependencies = c("Depends", "Imports", "Suggests", "LinkingTo"),
  recursive = FALSE,
  ignore = NULL,
  installed = NULL
)
```

Arguments

<code>pkg</code>	A character string representing the name of the package.
<code>dependencies</code>	A character vector specifying the types of dependencies to consider. Default is <code>c("Depends", "Imports", "Suggests", "LinkingTo")</code> .
<code>recursive</code>	A logical value indicating whether to find dependencies recursively. Default is <code>FALSE</code> .
<code>ignore</code>	A character vector of package names to ignore. Default is <code>NULL</code> .
<code>installed</code>	A matrix of installed packages.

Value

deps - A sorted character vector of package names that depend on the specified package.

Examples

```
## Not run:  
cran_revdep("ggplot2")  
  
## End(Not run)
```

create_coverage_skip_stf

Run covr with failing tests

Description

Run [package_coverage](#) ignoring failing tests from [test_dir](#).

Usage

```
create_coverage_skip_stf(test_path = "tests/testthat", test_skip, test_map)
```

Arguments

test_path	character, path to test directory, Default: 'tests/testthat'
test_skip	data.frame, mapping of failing tests, PARAM_DESCRIPTION,
test_map	data.frame, mapping of all the tests in the package

Details

If test_skip is NULL then a new test_dir will be run internally checking for failing tests.

See Also

[test_dir](#) [package_coverage](#)

Other utility: [map_test\(\)](#), [map_tests_stf\(\)](#)

`create_traceability_matrix`*Create a Traceability Matrix*

Description

Returns a table that links all exported functions and their aliases to their documentation ('man' files), the R scripts containing them, and the test scripts that reference them.

Usage

```
create_traceability_matrix(  
  pkg_name,  
  pkg_source_path,  
  func_covr = NULL,  
  verbose = FALSE,  
  execute_coverage = TRUE  
)
```

Arguments

<code>pkg_name</code>	name of package
<code>pkg_source_path</code>	path to a source package
<code>func_covr</code>	function coverage
<code>verbose</code>	Logical ('TRUE'/'FALSE'). If 'TRUE', show any warnings/messages per function.
<code>execute_coverage</code>	Logical ('TRUE'/'FALSE'). If 'TRUE', execute test coverage.

Value

a nested list with fine grained traceability matrices

Examples

```
## Not run:  
# set CRAN repo to enable running of reverse dependencies  
r = getOption("repos")  
r["CRAN"] = "http://cran.us.r-project.org"  
old <- options(repos = r)  
  
dp <- system.file("test-data", "here-1.0.1.tar.gz",  
  package = "risk.assessr")  
  
# Set up the package using the temporary file  
install_list <- set_up_pkg(dp)
```



```
options(old)

## End(Not run)
```

dependsOnPkgs *Determine Packages that Depend on Given Packages*

Description

This function identifies packages that depend on the specified packages, considering various types of dependencies (e.g., strong, most, all).

Usage

```
dependsOnPkgs(
  pkgs,
  dependencies = "most",
  recursive = TRUE,
  lib.loc = NULL,
  installed = NULL
)
```

Arguments

pkgs	A character vector of package names to check dependencies for.
dependencies	A character string specifying the types of dependencies to consider. Can be "strong", "most", "all", or a custom vector of dependency types.
recursive	A logical value indicating whether to recursively check dependencies.
lib.loc	A character vector of library locations to search for installed packages.
installed	A matrix of installed packages, obtained from cran_packages function. .

Value

A character vector of package names that depend on the specified packages.

Examples

```
## Not run:
installed <- cran_packages()
dependsOnPkgs("here", installed = installed)

## End(Not run)
```

`detect_version_conflicts`*Detect Version Conflicts from dependency tree*

Description

This function identifies duplicate version packages and reports conflicts.

Usage

```
detect_version_conflicts(pkg_list)
```

Arguments

`pkg_list` A nested list structure from `fetch_all_dependencies` function

Value

A list of strings describing package version conflicts such as "Conflict in package cli: versions found - 3.6.2, 3.6.1" 'NULL' if no conflicts are found.

Examples

```
pkg_list <- list(  
  ggplot2 = list(  
    version = "3.5.1",  
    cli = list(version = "3.6.2"),  
    gtable = list(  
      version = "0.3.4",  
      cli = list(version = "3.6.1")  
    )  
  )  
)  
detect_version_conflicts(pkg_list)
```

`download_and_parse_dependencies`*Download and Parse Dependencies of an R Package*

Description

Downloads a package from a repository, extracts it, and parses dependencies from the DESCRIPTION file.

Usage

```
download_and_parse_dependencies(  
  package_name,  
  version = NA,  
  get_license = FALSE  
)
```

Arguments

package_name	Name of the package to download.
version	Package version to download (default: 'NA' for latest).
get_license	Whether to extract license information (default: 'FALSE').

Details

The 'license' field refers only to the package specified by 'package_name', not its dependencies. Use 'build_dependency_tree()' or 'fetch_all_dependencies()' with 'get_license = TRUE' for full license information.

Value

When 'get_license = FALSE', a data frame with columns 'package', 'type', and 'parent_package'. When 'get_license = TRUE', a list with: - 'dependencies': Data frame of direct dependencies. - 'license': License of the specified package only (not dependencies). Returns 'NA' if not found.

Examples

```
## Not run:  
download_and_parse_dependencies("dplyr")  
download_and_parse_dependencies("dplyr", get_license = TRUE)  
  
## End(Not run)
```

extract_package_version

Extract the Installed Version of a Package

Description

This function retrieves the installed version of a specified R package. If the package is not installed, returns 'NA'.

Usage

```
extract_package_version(package_name)
```

Arguments

package_name A character string specifying the name of the package.

Value

A character string representing the installed package version, or 'NA' if the package is not found.

Examples

```
## Not run:  
extract_package_version("dplyr")  
# 1.1.4  
  
## End(Not run)
```

`extract_thresholds_by_id`
Extract risk thresholds by id

Description

Extract risk thresholds by id

Usage

```
extract_thresholds_by_id(risk_rule, id_value)
```

Arguments

risk_rule - list containing all risk rules
id_value - character vector containing risk rule choice

Value

list containing specific risk rules

extract_thresholds_by_key
Extract risk thresholds by key

Description

Extract risk thresholds by key

Usage

```
extract_thresholds_by_key(risk_rule, key_value)
```

Arguments

risk_rule - list containing all risk rules
key_value - character vector containing risk rule choice

Value

list containing specific risk rules

extract_version *Extract Package Version from File Path*

Description

This function extracts the version number from a package source file name based on the package name and expected file pattern.

Usage

```
extract_version(path, package_name)
```

Arguments

path A character string specifying the file path or URL.
package_name A character string specifying the name of the package.

Value

A character string representing the extracted version number, or 'NULL' if no match is found.

Examples

```
## Not run:
link <- "https://bioconductor.org/packages/3.14/bioc/src/contrib/GenomicRanges_1.42.0.tar.gz"
extract_version(link, "GenomicRanges")

## End(Not run)
```

fetch_all_dependencies

Fetch All Dependencies for a Package

Description

This function builds and retrieves the full dependency tree for a given R package.

Usage

```
fetch_all_dependencies(package_name, get_license = FALSE, max_level = 3)
```

Arguments

package_name	A character string specifying the name of the package.
get_license	A logical value indicating whether to extract and include license information from DESCRIPTION files (default: 'FALSE').
max_level	An integer specifying the maximum depth level for dependency recursion (default: 3). Set to a higher value to explore deeper dependency layers.

Value

A nested list representing the dependency tree of the package. If 'get_license' is 'TRUE', each package in the tree (including the root package and all dependencies) will have a 'license' field containing that package's license extracted from its DESCRIPTION file.

Examples

```
## Not run:
fetch_all_dependencies("ggplot2")
fetch_all_dependencies("ggplot2", get_license = TRUE)
fetch_all_dependencies("ggplot2", max_level = 5) # Deeper dependency exploration

## End(Not run)
```

`fetch_bioconductor_package_info`*Fetch Bioconductor Package Information*

Description

This function retrieves information about a specific Bioconductor package for a given Bioconductor version. It fetches the package details, such as version, source package URL, and archived versions if available.

Usage

```
fetch_bioconductor_package_info(bioconductor_version, package_name)
```

Arguments

`bioconductor_version`

A character string specifying the Bioconductor version (e.g., "3.14").

`package_name`

A character string specifying the name of the package.

Value

A list containing package details, including the latest version, package URL, source package link, and any archived versions if available. Returns 'FALSE' if the package does not exist or cannot be retrieved.

Examples

```
## Not run:
info <- fetch_bioconductor_package_info("3.14", "GenomicRanges")
print(info)

## End(Not run)
```

`fetch_bioconductor_releases`*Fetch Bioconductor Release Announcements*

Description

This function retrieves the Bioconductor release announcements page and returns its HTML content for further processing.

Usage

```
fetch_bioconductor_releases()
```

Value

An XML document from bioconductor version page.

Examples

```
## Not run:  
html_content <- fetch_bioconductor_releases()  
  
## End(Not run)
```

generate_html_report *Generate HTML Report for Package Assessment*

Description

Generates an HTML report for the package assessment results using rmarkdown.

Usage

```
generate_html_report(assessment_results, output_dir = NULL)
```

Arguments

```
assessment_results  
                List containing the results from risk_assess_pkg function.  
  
output_dir      Character string indicating the directory where the report will be saved.
```

Value

Path to the generated HTML report.

Examples

```
## Not run:  
assessment_results <- risk_assess_pkg()  
generate_html_report(assessment_results, output_dir = "path/to/save/report")  
  
## End(Not run)
```

`generate_traceability_matrix`

Assess an R Package traceability matrix from package name and version

Description

This function use ‘risk.assessr::create_traceability_matrix’ function with only the package name and version

Usage

```
generate_traceability_matrix(  
  package_name,  
  version = NA,  
  repos = getOption("repos"),  
  execute_coverage = FALSE  
)
```

Arguments

<code>package_name</code>	A character string specifying the name of the package to assess.
<code>version</code>	A character string specifying the version of the package to assess. Default is ‘NA’, which assesses the latest version.
<code>repos</code>	A character string specifying the repo directly. Default is <code>getOption("repos")</code>
<code>execute_coverage</code>	Logical (‘TRUE’/‘FALSE’). If ‘TRUE’, execute test coverage.

Value

The function returns package traceability_matrix If the package cannot be downloaded or installed, an error message is returned.

Examples

```
## Not run:  
  
results_no_test_covr <- generate_traceability_matrix(  
  "here",  
  version = "1.0.1",  
  execute_coverage = FALSE  
)  
  
results_test_covr <- generate_traceability_matrix(  
  "here",  
  version = "1.0.1",  
  execute_coverage = TRUE  
)
```

```
print(results_no_test_covr)

print(results_test_covr)

## End(Not run)
```

```
get_bioconductor_package_url
      Retrieve Bioconductor Package URL
```

Description

This function fetches the source package URL for a given Bioconductor package. If no version is specified, it retrieves the latest available version.

Usage

```
get_bioconductor_package_url(
  package_name,
  package_version = NULL,
  release_data
)
```

Arguments

`package_name` A character string specifying the name of the Bioconductor package.

`package_version` (Optional) A character string specifying the package version. Defaults to 'NULL', which retrieves the latest version.

`release_data` A list containing Bioconductor release information.

Value

A list containing:

`url` The URL of the source package (if available).

`version` The specified or latest available package version.

`last_version` A list with version, date, and Bioconductor version.

`all_versions` A list of all discovered versions with version, date, and Bioconductor version.

`bioconductor_version_package` The Bioconductor version matched to the selected version.

`archived` A logical value indicating whether the package is archived.

Examples

```
## Not run:
release_data <- list(
  list(release = "3.17", date = "October 14, 2005"),
  list(release = "3.18", date = "October 4, 2006"),
  list(release = "3.19", date = "October 8, 2007")
)

get_bioconductor_package_url("GenomicRanges", release_data = release_data)

## End(Not run)
```

get_commits_since *Retrieve GitHub Commits as Weekly Counts (using curl)*

Description

This function fetches all commits from a specified GitHub repository using the GitHub API and returns a data frame with weekly commit counts.

Usage

```
get_commits_since(owner, repo, years = 1)
```

Arguments

owner	Character. The GitHub username or organization that owns the repository.
repo	Character. The name of the GitHub repository.
years	Numeric. Number of years to look back from today's date (default is 1).

Value

A data frame with four columns:

week_start Start date of the week (class Date).

year Year of the commits.

month Month of the commits.

n_commits Number of commits during the week.

Examples

```
## Not run:
# Get commit counts for the past year
get_commits_since(owner = "tidyverse", repo = "ggplot2")

## End(Not run)
```

`get_cran_package_url` *Get CRAN Package URL*

Description

This function constructs the CRAN package URL for a specified package and version.

Usage

```
get_cran_package_url(package_name, version, last_version, all_versions)
```

Arguments

<code>package_name</code>	A character string specifying the name of the package.
<code>version</code>	An optional character string specifying the version of the package. If 'NULL', the latest version is assumed.
<code>last_version</code>	A named list with elements <code>version</code> and <code>date</code> , representing the latest available version and its publication date.
<code>all_versions</code>	A list of named lists, each with elements <code>version</code> and <code>date</code> , representing all known versions of the package and their publication dates.

Value

A character string containing the URL to download the package tarball, or "No valid URL found" if the version is not found in the list of available versions.

Examples

```
## Not run:  
# Example data structure  
last_version <- list(version = "1.0.10", date = "2024-01-01")  
all_versions <- list(  
  list(version = "1.0.0", date = "2023-01-01"),  
  list(version = "1.0.10", date = "2024-01-01")  
)  
  
# Get the URL for the latest version of "dplyr"  
url <- get_cran_package_url("dplyr", NULL, last_version, all_versions)  
print(url)  
  
## End(Not run)
```

`get_cran_total_downloads`*Get CRAN Total or Recent Downloads for a Package*

Description

Retrieves either all-time total downloads or the total from the last ‘n’ full months for a given CRAN package.

Usage

```
get_cran_total_downloads(pkg, months = NULL)
```

Arguments

<code>pkg</code>	Character string, the name of the CRAN package.
<code>months</code>	Integer. If NULL, returns all-time total; otherwise, returns total from the last ‘months’ full months (excluding current).

Details

Similar data can be obtained via:

<https://cranlogs.r-pkg.org/badges/grand-total/ggplot2> <https://cranlogs.r-pkg.org/badges/last-month/ggplot2>

Value

An integer: total number of downloads.

Examples

```
## Not run:
  get_cran_total_downloads("ggplot2")      # all-time total
  # 160776616

  get_cran_total_downloads("ggplot2", 3)   # last 3 full months
  # 4741707

  get_cran_total_downloads("ggplot2", 12)  # last 12 full months
  # 19878885

## End(Not run)
```

get_exports	<i>list all package exports</i>
-------------	---------------------------------

Description

list all package exports

Usage

```
get_exports(pkg_source_path)
```

Arguments

pkg_source_path
a file path pointing to an unpacked/untarred package directory

Value

data.frame, with one column 'exported_function', that can be passed to all downstream map_* helpers

Examples

```
## Not run:
tmpdir <- tempdir()

# Locate the package source tarball bundled with risk.assessr
dp <- system.file("test-data", "test.package.0001_0.1.0.tar.gz",
                  package = "risk.assessr")

# Extract the source package into the temporary directory
utils::untar(dp, exdir = tmpdir)

# Find the extracted package directory (it should contain DESCRIPTION)
pkg_dirs <- list.dirs(tmpdir, full.names = TRUE, recursive = FALSE)
desc_dir <- pkg_dirs[file.exists(file.path(pkg_dirs, "DESCRIPTION"))]
stopifnot(length(desc_dir) == 1L)
pkg_source_path <- desc_dir

# Now operate on the *source* (no installation)
exports_df <- get_exports(pkg_source_path)

# Optional: inspect result (kept simple for examples)
head(exports_df)

# Clean up the extracted sources
unlink(pkg_source_path, recursive = TRUE, force = TRUE)

## End(Not run)
```

get_github_data	<i>Fetch GitHub Repository Data</i>
-----------------	-------------------------------------

Description

This function retrieves metadata about a GitHub repository, including creation date, stars, forks, and the number of recent commits within the last 30 days.

Usage

```
get_github_data(owner, repo)
```

Arguments

owner	A character string specifying the owner of the repository.
repo	A character string specifying the name of the repository. A GitHub Personal Access Token (PAT) will be needed for some requests or to help with the rate limit. Use <code>Sys.setenv(GITHUB_TOKEN = "personal_access_token")</code> or store your token in a <code>.Renviro</code> n file. The token is passed via the <code>'Authorization'</code> header only if <code>'Sys.getenv("GITHUB_TOKEN")'</code> is non-empty.

Details

Repository data is fetched using the GitHub API via `'curl'` and `'jsonlite'`.

Value

A list containing: - `'created_at'`: Creation date of the repository. - `'stars'`: Number of stars. - `'forks'`: Number of forks. - `'date'`: Acquisition date (YYYY-MM-DD). - `'recent_commits_count'`: Number of commits in the last 30 days. - `'open_issues'`: Number of open issues.

Examples

```
## Not run:  
result <- get_github_data("tidyverse", "ggplot2")  
print(result)  
  
## End(Not run)
```

get_host_package	<i>Extract and Validate Package Hosting Information</i>
------------------	---

Description

This function retrieves hosting links for an R package from various sources such as GitHub, CRAN, internal repositories, or Bioconductor.

Usage

```
get_host_package(pkg_name, pkg_version, pkg_source_path)
```

Arguments

pkg_name	Character. The name of the package.
pkg_version	Character. The version of the package.
pkg_source_path	Character. The file path to the package source directory containing the DESCRIPTION file.

Details

The function extracts hosting links by: 1. Parsing the 'DESCRIPTION' file for GitHub and BugReports URLs. 2. Checking if the package is valid on CRAN and others host repository

If no links are found in the 'DESCRIPTION' file, returns 'NULL'

Value

A list containing the following elements:

- 'github_links': GitHub links related to the package. - 'cran_links': CRAN links - 'internal_links': internal repository links. - 'bioconductor_links': Bioconductor links

If links are found, return empty or NULL.

Examples

```
## Not run:  
result <- get_host_package(pkg_name, pkg_version, pkg_source_path)  
print(result)  
  
## End(Not run)
```

`get_internal_package_url`*Get Internal Package URL*

Description

This function retrieves the URL of an internal package on Sanofi Mirror, its latest version, and a list of all available versions.

Usage

```
get_internal_package_url(package_name, version = NULL, base_url = NULL)
```

Arguments

`package_name` A character string specifying the name of the package.
`version` An optional character string specifying the version of the package. Defaults to 'NULL', in which case the latest version will be used.
`base_url` A character string specifying the base URL of the internal package manager.

Value

A list containing: - 'url': A character string of the package URL (or 'NULL' if not found). - 'last_version': A list with 'version' and 'date' of the latest version (or 'NULL'). - 'version': The version used to generate the URL (or 'NULL'). - 'all_versions': A list of all available versions, each as a list with 'version' and 'date'.

Examples

```
## Not run:  
result <- get_internal_package_url("internalpackage", version = "1.0.1")  
print(result)  
  
## End(Not run)
```

`get_package_download_bioconductor`*Get Bioconductor Package Download Statistics*

Description

Downloads and processes monthly download statistics for a Bioconductor package from the Bioconductor package statistics archive. Returns the full history, the total downloads over the last 6 full months (current month excluded), and the all-time total downloads.

Usage

```
get_package_download_bioconductor(pkg)
```

Arguments

pkg Character string. The name of a Bioconductor package (e.g. "GenomicRanges").

Value

A list with three elements:

all_data A data.frame of the full cleaned dataset with dates and cumulative counts.

total_6_months Total downloads in the last 6 complete months.

total_download Total all-time downloads.

Examples

```
## Not run:
result <- get_package_download_bioconductor("bioconductorpackage")
print(result$total_6_months)
head(result$all_data)

#$all_data
#Date Year Month Nb_of_distinct_IPs Nb_of_downloads Cumulative_downloads
#1 2009-01-01 2009 Jan 3341 7053 7053
#2 2009-02-01 2009 Feb 3229 6681 13734
#3 2009-03-01 2009 Mar 3753 8021 21755

#$total_6_months
#[1] 547560

#$total_download
#[1] 7318550

## End(Not run)
```

```
get_package_download_cran
```

Get CRAN Daily Downloads for a Package

Description

Retrieves daily CRAN download counts over the past years for a given package, including cumulative downloads.

Usage

```
get_package_download_cran(pkg, years = 1)
```

Arguments

pkg Character string, name of the CRAN package.
 years integer, number of past years (default 1) for historical

Value

A data frame with columns: date, count, and cumulative_downloads.

Examples

```
## Not run:
get_package_download_cran("ggplot2")
# Example output:
#       date    count cumulative_downloads
# 1 2025-04-22  78379           20073615
# 2 2025-04-21  63195           19995236
# 3 2025-04-20  42119           19932041
# 4 2025-04-19  40848           19889922
# 5 2025-04-18  54914           19849074
# 6 2025-04-17  86273           19794160
# 7 2025-04-16  80201           19707887

## End(Not run)
```

get_package_tarfile *Download R Package Source Tarball*

Description

Downloads the source tarball (.tar.gz) for a specified R package from CRAN, Bioconductor, or an internal repository fallback.

Usage

```
get_package_tarfile(package_name, version = NULL, repos = getOption("repos"))
```

Arguments

package_name Character string specifying the name of the package to download.
 version Optional character string specifying the version of the package. If NULL (default), the latest available version will be downloaded.
 repos Optional character vector of repository URLs to use. If NULL (default), the current repositories set in options() will be used.

Value

Character string with the path to the downloaded tarball file, or NULL if the download was unsuccessful.

Examples

```
## Not run:
# Download the latest version of dplyr
tarball_path <- get_package_tarfile("dplyr")

# Download a specific version
tarball_path <- get_package_tarfile("dplyr", version = "1.0.0")

# Use a specific repository
tarball_path <- get_package_tarfile("dplyr", repos = "https://cloud.r-project.org")

## End(Not run)
```

get_pkg_name	<i>get package name for display</i>
--------------	-------------------------------------

Description

get package name for display

Usage

```
get_pkg_name(input_string)
```

Arguments

input_string - string containing package name

Value

pkg_disp - package name for display

Examples

```
pkg_source_path <- "/home/user/R/test.package.0001_0.1.0.tar.gz"
pkg_disp_1 <- get_pkg_name(pkg_source_path)
print(pkg_disp_1)

pkg <- "TxDb.Dmelanogaster.UCSC.dm3.ensGene_3.2.2.tar.gz"
pkg_disp_2 <- get_pkg_name(pkg)
print(pkg_disp_2)
```

get_pubmed_by_year *Get Annual PubMed Article Counts for a Search Term*

Description

This function queries the NCBI E-utilities API to retrieve the number of PubMed articles published each year over a specified number of past years for a given search term or package name.

Usage

```
get_pubmed_by_year(  
  package_name,  
  years_back = 10,  
  api_key = getOption("pubmed.api_key", NULL)  
)
```

Arguments

package_name A character string representing the search term (e.g., an R package name).
years_back An integer specifying how many years back to query. Defaults to 10.
api_key Optional. A character string with an NCBI API key. If not supplied, it will attempt to use the option `getOption("pubmed.api_key")`.

Value

A data frame with two columns:

Year The publication year (integer).

Count The number of articles published in that year (integer).

Returns an empty data frame if no valid data is retrieved.

Examples

```
## Not run:  
get_pubmed_by_year("ggplot2", years_back = 5)  
  
## End(Not run)
```

get_pubmed_count	<i>Get Total Number of PubMed Articles for a Search Term</i>
------------------	--

Description

This function queries the NCBI E-utilities API to retrieve the total number of PubMed articles that match a given search term or R package name.

Usage

```
get_pubmed_count(package_name, api_key = getOption("pubmed.api_key", NULL))
```

Arguments

package_name	A character string representing the search term (e.g., an R package name).
api_key	Optional. A character string with an NCBI API key. If not supplied, it will attempt to use the option <code>getOption("pubmed.api_key")</code> .

Value

An integer representing the total number of PubMed articles matching the search term, or NA if the request fails.

Examples

```
## Not run:  
get_pubmed_count("ggplot2")  
  
## End(Not run)
```

get_risk_analysis	<i>Get Risk Analysis</i>
-------------------	--------------------------

Description

Compute risk levels for the package metadata.

Usage

```
get_risk_analysis(data)
```

Arguments

data	The nested package metadata.
------	------------------------------

Value

A named list of computed risk levels.

Examples

```
# Minimal mock input for demonstration
mock_data <- list(
  dependencies_count = 5,
  later_version = 2,
  code_coverage = 0.75,
  last_month_download = 150000,
  license = "MIT",
  reverse_dependencies_count = 10,
  documentation_score = 2,
  has_examples = list(
    data = data.frame(
      function_name      = "somefunction",
      documentation_name = "No documentation found",
      documentation_location = 0,
      example            = "no Rd file",
      stringsAsFactors  = FALSE
    ),
    example_score = 0.20
  ),
  has_docs = list(
    data = data.frame(
      function_name      = "somefunction",
      stringsAsFactors  = FALSE
    ),
    has_docs_score = 0.30
  ),
  cmd_check = 0
)

get_risk_analysis(mock_data)
```

get_risk_definition *Get Risk Definition*

Description

Load risk definition config from options, JSON file, or fallback to bundled file.

Usage

```
get_risk_definition()
```

Value

A list of risk configuration items.

Examples

```
risk_rule <- get_risk_definition()
print(risk_rule)
```

```
get_session_dependencies
      Get Dependencies
```

Description

This function extracts the version information of imported and suggested packages for a given package from the current R session.

Usage

```
get_session_dependencies(deps_list)
```

Arguments

deps_list	A data frame containing the dependency information of the package (provided by calc_dependencies function)
-----------	--

Value

A list with two elements:

imports	A named list of packages in the "Imports" section along with their corresponding versions
suggests	A named list of packages in the "Suggests" section along with their corresponding versions

Examples

```
deps_list <- data.frame(
  package = c("dplyr", "ggplot2", "testthat", "knitr"),
  type = c("Imports", "Imports", "Suggests", "Suggests")
)
get_session_dependencies(deps_list)
```

`get_suggested_exp_funcs`*Function to get suggested exported functions*

Description

This function gets exported functions for all packages in the Suggests section of the target package's DESCRIPTION file

Usage

```
get_suggested_exp_funcs(data)
```

Arguments

`data` - all packages listed in the DESCRIPTION file

Value

- data with package names and exported functions

`get_versions`*Get Package Versions*

Description

This function retrieves all available versions of a package, including the latest version, by parsing the provided version table and querying the RStudio Package Manager.

Usage

```
get_versions(table, package_name)
```

Arguments

`table` A list of parsed package data (e.g., from `'parse_html_version()'`), where each element contains package details such as `'package_version'` and `'date'`.

`package_name` A character string specifying the name of the package to fetch versions for.

Value

A list with the following elements:

all_versions A list of named lists, each containing:

version The version number of the package.

date The associated publication date as a string.

last_version A named list containing the latest version of the package with:

version The latest version number.

date The publication date of the latest version.

May be NULL if the API call fails.

Examples

```
## Not run:
# Define the input table
table <- list(
  list(
    package_name = "here",
    package_version = "0.1",
    link = "here_0.1.tar.gz",
    date = "2017-05-28 08:13",
    size = "3.5K"
  ),
  list(
    package_name = "here",
    package_version = "1.0.0",
    link = "here_1.0.0.tar.gz",
    date = "2020-11-15 18:10",
    size = "32K"
  )
)

result <- get_versions(table, "here")
print(result)

## End(Not run)
```

install_package_local *Install package locally*

Description

Install package locally

Usage

```
install_package_local(pkg_source_path)
```

Arguments

pkg_source_path
- source path for install local

Value

logical - package_installed

Examples

```
## Not run:  
results <- install_package_local("pkg_source_path")  
print(results)  
  
## End(Not run)
```

is_base	<i>Check if a Package is a Base or Recommended R Package</i>
---------	--

Description

This function determines whether a given package is a base or recommended package using CRAN metadata.

Usage

```
is_base(pkg)
```

Arguments

pkg A character string representing the name of the package.

Value

A logical value: 'TRUE' if the package is a base or recommended package, 'FALSE' otherwise.

Examples

```
## Not run:  
is_base("stats")      # TRUE  
is_base("ggplot2")    # FALSE  
  
## End(Not run)
```

list_badges	<i>List badges image URLs from a local README</i>
-------------	---

Description

Scans a local README (Markdown) and returns badge image URLs.

Usage

```
list_badges(path)
```

Arguments

path Character scalar; path to a local README file (e.g., "README.md").

Value

data.frame with badge info

Examples

```
## Not run:
tmp <- tempfile(fileext = ".md")
writeLines(c(
  "# MyPkg",
  "![build](build-status.svg)",
  "![cov](coverage.svg)"
), tmp)
out <- list_badges(tmp)
print(out)

## End(Not run)
```

map_test	<i>Map the hierarchy structure of a testthat test file</i>
----------	--

Description

Query a test file to see what expectations are defined in it.

Usage

```
map_test(path)
```

Arguments

path character, path to file

Details

Return data.frame containing which expectations are in the test file by context/description/test/expectation/linerange

Value

data.frame

See Also

[getParseData](#)

Other utility: [create_coverage_skip_stf\(\)](#), [map_tests_stf\(\)](#)

map_tests_stf	<i>Map the hierarchy structure of tests directory in standard testing framework</i>
---------------	---

Description

Query a test directory for the unit test structure.

Usage

```
map_tests_stf(path = "tests/testthat")
```

Arguments

path character, path to tests, Default: 'tests/testthat'

Details

Return data.frame containing which expectations are in the testthat directory by file/context/description/test/expectation/linerange

Value

data.frame

See Also

[names](#), [lapply](#), [rbind](#)

Other utility: [create_coverage_skip_stf\(\)](#), [map_test\(\)](#)

`modify_description_file`*Modify the DESCRIPTION File in a R Package Tarball*

Description

This function recreate a ‘.tar.gz’ R package file after modifying its ‘DESCRIPTION’ file by appending `Config/build/clean-inst-doc: false` parameter.

Usage

```
modify_description_file(tar_file)
```

Arguments

`tar_file` A string representing the path to the ‘.tar.gz’ file that contains the R package.

Value

A string containing the path to the newly created modified ‘.tar.gz’ file.

Examples

```
## Not run:  
modified_tar <- modify_description_file("path/to/mypackage.tar.gz")  
  
## End(Not run)
```

`parse_bioconductor_releases`*Parse Bioconductor Release Announcements*

Description

This function extracts Bioconductor release details such as version number, release date, number of software packages, and required R version from the release announcements HTML page.

Usage

```
parse_bioconductor_releases(html_content)
```

Arguments

`html_content` The parsed HTML document from ‘fetch_bioconductor_releases’.

Value

A list of lists containing Bioconductor release details: release version, date, number of software packages, and corresponding R version.

Examples

```
## Not run:  
html_content <- fetch_bioconductor_releases()  
release_data <- parse_bioconductor_releases(html_content)  
print(release_data)  
  
## End(Not run)
```

parse_package_info *Parse Package Information from CRAN Archive*

Description

This function retrieves the package archive information from the CRAN Archive.

Usage

```
parse_package_info(name)
```

Arguments

name A character string specifying the name of the package to fetch information for.

Value

A character string containing the raw HTML content of the package archive page, or 'NULL' if the request fails or the package is not found.

Examples

```
## Not run:  
# Fetch package archive information for "dplyr"  
parse_package_info("dplyr")  
  
## End(Not run)
```

print_tree	<i>Print a Package Dependency Tree</i>
------------	--

Description

This function prints a hierarchical representation of a package's dependencies, including version and base status

Usage

```
print_tree(tree, prefix = "", last = TRUE, show_version = TRUE)
```

Arguments

tree	A nested list representing the package dependency tree, where each package has a 'version' field and potentially sub-dependencies.
prefix	A character string used for formatting tree branches (default: "").
last	A logical value indicating if the current package is the last in the tree level (default: 'TRUE').
show_version	A logical value indicating whether to display package versions if available (default: 'TRUE').

Value

Prints the dependency tree to the console.

Examples

```
## Not run:
db <- list(
  stringr = list(
    version = "1.5.1",
    cli = list(version = "3.6.2", utils = "base"),
    glue = list(version = "1.7.0", methods = "base"),
    lifecycle = list(
      version = "1.0.4",
      cli = list(version = "3.6.2", utils = "base"),
      glue = list(version = "1.7.0", methods = "base"),
      rlang = list(version = "1.1.3", utils = "base")
    ),
  magrittr = list(version = "2.0.3"),
  rlang = list(version = "1.1.3", utils = "base"),
  stringi = list(version = "1.8.3", tools = "base", utils = "base", stats = "base"),
  vctrs = list(
    version = "0.6.5",
    cli = list(version = "3.6.2", utils = "base"),
    glue = list(version = "1.7.0", methods = "base"),
    lifecycle = list(version = "1.0.4"),
```

```

    rlang = list(version = "1.1.3", utils = "base")
  )
)
print_tree(db)

## End(Not run)

```

risk_assess_pkg

Assess package for risk metrics

Description

Assess an R package for risk metrics. Use exactly one of: * ‘path’ – path to a local .tar.gz archive, or * ‘package’ – package name to fetch (optionally with ‘version’), or * neither – opens an interactive file chooser.

Usage

```

risk_assess_pkg(
  path = NULL,
  package = NULL,
  version = NA,
  repos = getOption("repos")
)

```

Arguments

path	Path to a local .tar.gz package archive. Use only when not providing ‘package’.
package	Package name to fetch from CRAN, Bioconductor, or internal repos. Use only when not providing ‘path’.
version	(optional) Package version when using ‘package’. Default ‘NA’ uses the latest available.
repos	Repository URLs when fetching by ‘package’. Default is ‘getOption("repos")’.

Value

List containing results (metrics, covr, trace matrix, R CMD check), or ‘NULL’ if the package cannot be assessed.

Examples

```

## Not run:
# Local package (path or file chooser)
results <- risk_assess_pkg()
results <- risk_assess_pkg(path = "path/to/package.tar.gz")

```

```
# Package by name from repositories
results <- risk_assess_pkg(package = "here")
results <- risk_assess_pkg(package = "here", version = "1.0.1")
results <- risk_assess_pkg(package = "here", repos = "https://cloud.r-project.org")

## End(Not run)
```

risk_assess_pkg_lock_files
Process lock files

Description

This function processes ‘renv.lock’ and ‘pak.lock’ files to produce risk metric data

Usage

```
risk_assess_pkg_lock_files(input_data)
```

Arguments

input_data - path to a lock file

Value

assessment_results - nested list containing risk metric data

Examples

```
## Not run:
input_data <- ("path/to/mypak.lock")
pak_results <- risk_assess_pkg_lock_files(input_data)
print(pak_results)

## End(Not run)
```

run_covr_modes *Run Coverage Analysis with Test Detection*

Description

This function checks the test configuration of an R package and runs code coverage analysis using ‘testthat’, ‘testit’ or ‘test base’ frameworks if available. If no supported test framework is found, it returns a default coverage result indicating no functions were tested.

Usage

```
run_covr_modes(pkg_source_path, covr_timeout = 60)
```

Arguments

pkg_source_path Character. Path to the root directory of the R package source.

covr_timeout Numeric. Timeout in seconds for running coverage analysis. Default is 60.

Value

A list with two elements:

covr_list A list containing coverage results. If no supported tests are found, returns a default structure with zero coverage.

test_pkg_data A list summarizing the test configuration, including presence of ‘testthat’, ‘testit’, snapshots, and base R test scripts.

Examples

```
## Not run:
r = getOption("repos")
r["CRAN"] = "http://cran.us.r-project.org"
old <- options(repos = r)

dp <- system.file("test-data",
  "test.package.0001_0.1.0.tar.gz",
  package = "risk.assessr")

# set up package
install_list <- set_up_pkg(dp)

build_vignettes <- install_list$build_vignettes
package_installed <- install_list$package_installed
pkg_source_path <- install_list$pkg_source_path
rcmdcheck_args <- install_list$rcmdcheck_args

# install package locally to ensure test works
package_installed <- install_package_local(pkg_source_path)
package_installed <- TRUE

rcmdcheck_args$path <- pkg_source_path
covr_mode_list <- run_covr_modes(pkg_source_path)

## End(Not run)
```

run_covr_skip_nstf *Run Coverage Analysis on Package Source and Tests*

Description

This internal function maps source files to test files, checks for skipped/problematic tests, runs the remaining test files manually, calculates coverage, and optionally generates a report.

Usage

```
run_covr_skip_nstf(pkg_source_path, test_pkg_data)
```

Arguments

`pkg_source_path` Character string. Path to the root of the package source directory.

`test_pkg_data` List. Output from `'check_pkg_tests_and_snaps()'` indicating test framework presence.

Value

returns a list with coverage object and source files with no tests.

Examples

```
## Not run:
r = getOption("repos")
r["CRAN"] = "http://cran.us.r-project.org"
options(repos = r)

dp <- system.file("test-data",
  "MASS_7.3-65.tar.gz",
  package = "risk.assessr")

# set up package
install_list <- set_up_pkg(dp)

build_vignettes <- install_list$build_vignettes
package_installed <- install_list$package_installed
pkg_source_path <- install_list$pkg_source_path
rcmdcheck_args <- install_list$rcmdcheck_args

# install package locally to ensure test works
package_installed <- install_package_local(pkg_source_path)
package_installed <- TRUE

rcmdcheck_args$path <- pkg_source_path
test_pkg_data <- check_pkg_tests_and_snaps(pkg_source_path)
test_covr_list <- run_covr_skip_nstf(pkg_source_path, test_pkg_data)
```

```
options(old)

## End(Not run)
```

```
run_covr_skip_stf      Run covr with failing tests #' This function runs test coverage and skip
                        analysis for a package using testthat or other supported frameworks.
```

Description

Run covr with failing tests #' This function runs test coverage and skip analysis for a package using testthat or other supported frameworks.

Usage

```
run_covr_skip_stf(pkg_source_path, test_pkg_data)
```

Arguments

```
pkg_source_path      Character. Path to the root of the package source.

test_pkg_data        List. Output from 'check_pkg_tests_and_snaps()' indicating test framework
                        presence.
```

Value

skip_results - test coverage with PASS tests passing or FAIL for tests failing

Examples

```
## Not run:
r = getOption("repos")
r["CRAN"] = "http://cran.us.r-project.org"
old <- options(repos = r)

dp <- system.file("test-data",
  "test.package.0014_0.1.0.tar.gz",
  package = "risk.assessr")

# set up package
install_list <- set_up_pkg(dp)

build_vignettes <- install_list$build_vignettes
package_installed <- install_list$package_installed
pkg_source_path <- install_list$pkg_source_path
rcmdcheck_args <- install_list$rcmdcheck_args

# install package locally to ensure test works
```

```
package_installed <- install_package_local(pkg_source_path)
package_installed <- TRUE

rcmdcheck_args$path <- pkg_source_path
test_pkg_data <- check_pkg_tests_and_snaps(pkg_source_path)
test_covr_list <- run_covr_skip_stf(pkg_source_path, test_pkg_data)

options(old)

## End(Not run)
```

set_up_pkg

Creates information on package installation

Description

Creates information on package installation

Usage

```
set_up_pkg(dp, check_type = "1")
```

Arguments

dp data path and name for the package.
check_type basic R CMD check type - "1" CRAN R CMD check_type - "2"

Value

list with local package install

Examples

```
## Not run:
set_up_pkg(path/to/package, "mypackage")

## End(Not run)
```

write_summary_report *Write the risk assessment summary report*

Description

Renders the package risk assessment report ('summary.Rmd') to a chosen format and writes it to disk. The output file path can be inferred or explicitly set. You can specify the output format directly via 'output_format' or let it be inferred from the extension of 'output_file'. If 'output_dir' is not provided, the current working directory is used.

Usage

```
write_summary_report(
  results,
  output_file = NULL,
  output_format = NULL,
  output_dir = NULL
)
```

Arguments

results	A results object containing the risk assessment results. It is expected to include 'results\$results\$pkg_name' and 'results\$results\$pkg_version', which are used to construct default file names and report parameters.
output_file	'character(1)' or 'NULL'. The output file path **or** a directory. - If 'NULL', a default name is generated: "summary_report_<pkg_name>_<pkg_version>.html" in 'output_dir'. - If it is a directory, the default name is composed inside it. - If it is a relative path, it is resolved against 'output_dir'.
output_format	'character(1)' or 'NULL'. Explicit output format to render: one of "html" or "md" (case-insensitive; do **not** include a leading dot). If 'NULL', the format is inferred from the extension of 'output_file'. Unknown values trigger a warning and default to HTML.
output_dir	'character(1)' or 'NULL'. Base directory used when 'output_file' is 'NULL' or a relative path. If 'NULL', defaults to the current working directory ('getwd()'), with an informational message.

Details

The function calls [rmarkdown::render()] on an internal template: 'system.file("report_templates", "summary.Rmd", package = "risk.assess")'.

The format mapping is: - "html" -> [rmarkdown::html_document()] - "md" -> [rmarkdown::md_document()]

If 'output_format' is provided, it takes precedence for rendering even if 'output_file' has a different extension; in that case the filename is adjusted to match the format. If 'output_format' is not provided, the function infers the format from 'output_file's extension. For unknown extensions, it warns and rewrites the extension to '.html'.

Value

(Invisibly) the absolute path to the written report file.

Behavior

- If 'output_dir' is 'NULL', it defaults to 'getwd()' and a message is printed. - If 'output_file' is 'NULL', a file name is auto-generated as "summary_report_<pkg>_<version>.html" inside 'output_dir'. - If 'output_file' points to an existing directory (or ends with a path separator), a file name is composed inside that directory as "summary_report_<pkg>_<version>.html". - If 'output_file' is a relative path, it is resolved against 'output_dir'. - If 'output_format' is provided, it determines the render format (and the 'output_file' name is adjusted to match the format). - If 'output_format' is not provided, the render format is inferred from the extension of 'output_file'. Unknown extensions trigger a warning and the extension is changed to '.html'.

Examples

```
## Not run:
# generate assessment_results
results <- risk_assess_pkg()

# Basic usage: auto-name in the working directory (HTML)
write_summary_report(results)

# Specify output directory (auto-name inside that directory)
write_summary_report(results, output_dir = "artifacts")

# Provide an explicit filename (relative to output_dir)
write_summary_report(results, output_file = "reports/summary.html")

# Provide a directory as output_file: auto-name inside that directory
write_summary_report(results, output_file = "reports/")

# Render to Markdown explicitly
write_summary_report(results, output_format = "md")

# Let the extension drive the format (md)
write_summary_report(results, output_file = "reports/summary.md")

# Case-insensitive formats work; avoid leading dots (".md" is not valid)
write_summary_report(results, output_format = "MD")

## End(Not run)
```

Index

* utility

- create_coverage_skip_stf, [13](#)
- map_test, [43](#)
- map_tests_stf, [44](#)

- assess_pkg_r_package, [3](#)
- average_issue_close_time, [4](#)

- bioconductor_reverse_deps, [5](#)
- build_dependency_tree, [6](#)

- check_and_fetch_cran_package, [7](#)
- check_cran_package, [8](#)
- check_forbidden_notes, [8](#)
- check_suggested_exp_funcs, [9](#)
- contains_vignette_folder, [10](#)
- count_commits_last_months, [11](#)
- cran_packages, [11](#)
- cran_revdep, [12](#)
- create_coverage_skip_stf, [13](#), [44](#)
- create_traceability_matrix, [14](#)

- dependsOnPkgs, [16](#)
- detect_version_conflicts, [17](#)
- download_and_parse_dependencies, [17](#)

- extract_package_version, [18](#)
- extract_thresholds_by_id, [19](#)
- extract_thresholds_by_key, [20](#)
- extract_version, [20](#)

- fetch_all_dependencies, [21](#)
- fetch_bioconductor_package_info, [22](#)
- fetch_bioconductor_releases, [22](#)

- generate_html_report, [23](#)
- generate_traceability_matrix, [24](#)
- get_bioconductor_package_url, [25](#)
- get_commits_since, [26](#)
- get_cran_package_url, [27](#)
- get_cran_total_downloads, [28](#)

- get_exports, [29](#)
- get_github_data, [30](#)
- get_host_package, [31](#)
- get_internal_package_url, [32](#)
- get_package_download_bioconductor, [32](#)
- get_package_download_cran, [33](#)
- get_package_tarfile, [34](#)
- get_pkg_name, [35](#)
- get_pubmed_by_year, [36](#)
- get_pubmed_count, [37](#)
- get_risk_analysis, [37](#)
- get_risk_definition, [38](#)
- get_session_dependencies, [39](#)
- get_suggested_exp_funcs, [40](#)
- get_versions, [40](#)
- getParseData, [44](#)

- install_package_local, [41](#)
- is_base, [42](#)

- lapply, [44](#)
- list_badges, [43](#)

- map_test, [13](#), [43](#), [44](#)
- map_tests_stf, [13](#), [44](#), [44](#)
- modify_description_file, [45](#)

- names, [44](#)

- package_coverage, [13](#)
- parse_bioconductor_releases, [45](#)
- parse_package_info, [46](#)
- print_tree, [47](#)

- rbind, [44](#)
- risk_assess_pkg, [48](#)
- risk_assess_pkg_lock_files, [49](#)
- run_covr_modes, [49](#)
- run_covr_skip_nstf, [51](#)
- run_covr_skip_stf, [52](#)

set_up_pkg, [53](#)

test_dir, [13](#)

write_summary_report, [54](#)